# File Level Provenance Tracking in CMS

**C D Jones[1], J Kowalkowski[1], M Paterno[1], L Sexton-Kennedy[1],**

**W Tanenbaum[1] and D S Riley[2]**

[1]Fermi National Accelerator Laboratory, P.O. Box 500, Batavia IL 60510-5011, USA
[2]Wilson Laboratory, Cornell University, Ithaca NY 14853-8001, USA

E-mail: cdj@fnal.gov

**Abstract**. The CMS off-line framework stores provenance information within CMS's standard ROOT event data files. The provenance information is used to track how each data product was constructed, including what other data products were read to do the construction. We will present how the framework gathers the provenance information, the efforts necessary to minimise the space used to store the provenance in the file and the tools that will be available to use the provenance.

## 1. Introduction

The LHC experiments are anticipated to acquire unprecedently large HEP data samples over their lifetime. With the possibility of frequent reprocessing of the data, especially over the first few years, the experiments need to record detailed information to understand the history of how the data were chosen and produced. Such provenance information does not have to be sufficient to allow an exact replay of a process. It only needs to help understand how the data were created. Although provenance can be stored externally from the actual data files, the existence of provenance information in the data files is important for the large scale, highly distributed production to insure trust in the data. This is especially true for physicist's personal skims, which are not centrally managed by CMS.

### 1. 1. CMS processing model

What process level provenance can be recorded is defined by the processing model of the application. CMS's off-line processing framework [1] composes separate modules into groups. These groups do operations using an Event, as illustrated in Figure 1. The processing framework is composed of the following concepts

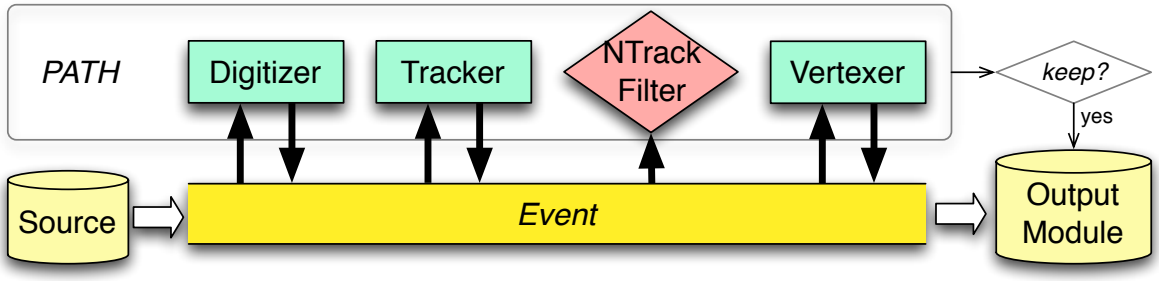


**Figure 1**. Simple representation of the CMS data processing framework.

- **Event**: An Event is a container of data associated with one HEP event.
- **Sources**: Sources are modules that create Events and read event data from an external source that was created by a previous processing step.

- **Producers**: Producers are modules that read data from an Event and put new data products into the Event.
- **Filters**: Filters are modules that read data from an Event and decide if the Event passes some criteria. Filters may optionally put data products into the Event to record why they made a decision.
- **Paths**: Paths are ordered set of Producers and Filters, where Filter decisions can cause a Path to stop processing an event.
- **OutputModules**: OutputModules are modules that look at the success or failure of the Paths to decide if data from an event should be saved.

Figure 1 illustrates the event data processing. The source creates an Event and places event data within the Event. The Event is then passed to a Path. This Path then sequentially passes the Event to each module in the Path. In the above example, the Digitizer is called first followed by the Tracker and then the NTrackFilter. Based on the return value of a Filter, illustrated by NTrackFilter in figure 1, the Path may continue to the next module or it may stop processing the event. Once all Paths have completed, the OutputModules are called. The decision on whether the event data are written to the output file is based on the success or failure of a logical 'or'ing of selected Paths.

### 1. 2. Provenance types

Provenance can be gathered externally and internally to an application. External provenance records which files were read in a process (both data and configuration) as well as what applications were used to generate a group of files. For example, external provenance might say that file *DEF.root* was made by the application *cmsRun* by reading the file *ABC.root* and configuration file *prod.py*. External provenance is recorded in CMS by the workflow management system [2].

Internal provenance records the internal state of the application. For CMS, this information is stored in the output file along with the event data. Because of the way the CMS processing framework operates, we need to answer provenance questions at three different levels: per event level, per data products level and per event per data product level. The per event provenance can answer what filters were applied to choose the stored events (e.g., event filters MaxTracks and MinJets had to pass an event for it to be stored) and what happened in the application while the event was being processed (e.g. module FooBar threw an exception but the framework ignored it and kept processing). The per data product provenance answers the questions what Producers created which data products (e.g., Producer JetFinder creates a std::vector<Jet> with label 'jets') and how the Producers of each data product were configured (e.g. Producer JetFinder had the parameter 'threshold' set to 5). Finally the per event per data product provenance answers what data products were read by a Producer to create the new product, e.g., in event 10 the JetFinder read the list of calorimeter towers.

## 2. Recording of the Provenance

The CMS processing framework records provenance during job startup as well as for each event. Exactly what is stored at each stage is dependent upon what items the framework allows to be changed at which stage.

### 2. 1. Startup

During startup, each module and Path is constructed. Once constructed it cannot be modified. Each module is passed its configuration information (i.e. parameters) when it is constructed. Also during construction, each module registers what data it will produce. Because a module or Path cannot be modified, its provenance is recorded at startup. In particular we record the Path and module configurations for this job, the software version being used, and what Paths the OutputModule that is writing this output file is monitoring.

2. 2. Event

The processing framework records several different kinds of provenance for each event. First the framework records information for each processing step (e.g., HLT and RECO) that contributed data to this event. The framework records both the order in which the processing steps were run (e.g., HLT before RECO) and the configuration used for each step. This information is copied from the Source to the Event to eventually be recorded by the OutputModule. When the framework executes a Producer for an event, the framework monitors each data product requested through the Event by the Producer. If the Producer finishes successfully (i.e. it does not throw an exception), then the framework publishes the data products constructed by the Producer and records what data were requested. Both pieces of information will be written to the output file if the event is chosen. For Paths, the framework records whether or not the Path ran to completion. If a Path fails, the framework records the last module executed and why it stopped the Path (e.g., because the filter rejected the event or because the module threw an exception). When all Paths are finished, the framework aggregates the results for all Paths into one data product and publishes that data product to the Event. OutputModules, which are run only after all Paths have finished, write out the provenance accumulated for the event.

2. 3. File format

A simplified schematic of the provenance information stored in the ROOT [3] file format used by CMS is shown in figure 2. The decision of whether to record information per event or just once per file is based on how often a given piece of information changes. Because per event data is often repeated,

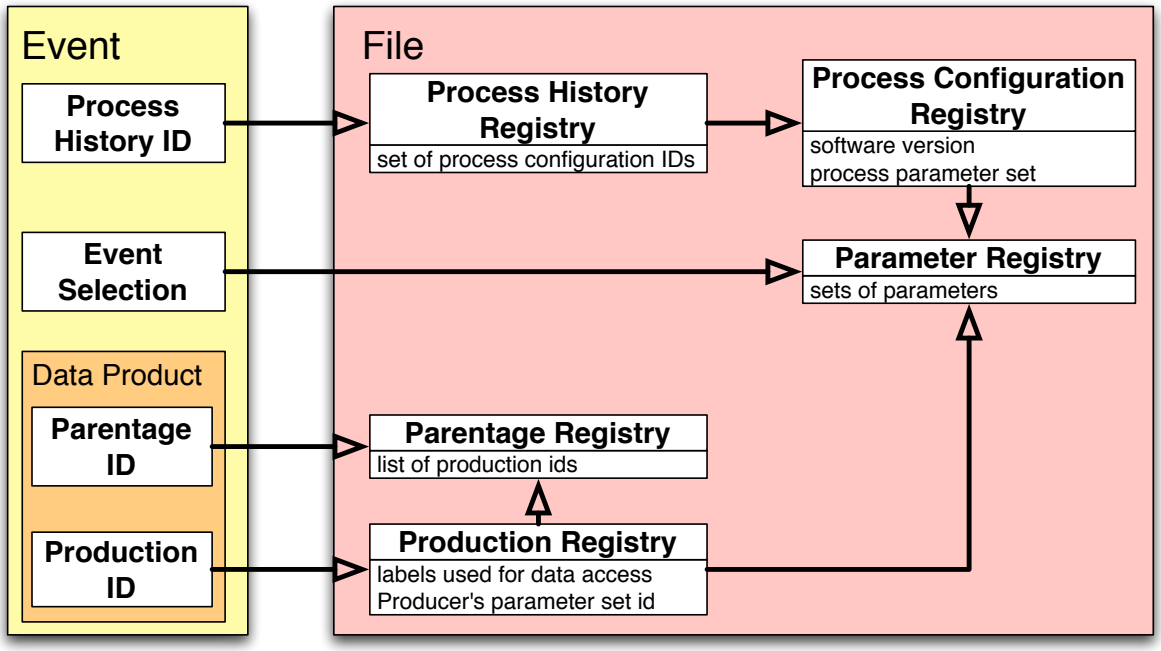


**Figure 2**. Schematic of the provenance tracking portions of the CMS ROOT file format.

our strategy is to store the full information into a registry and save only an ID into that registry in each event.

All configuration information in the CMS processing framework is reduced to sets of parameters that are stored in the Parameter Registry. Each set of parameters is assigned its own ID. The Process Configuration Registry records information for each processing step that contributed to the history of

the file. For each processing step, it records the software version used in that step. It also records the ID in the Parameter Registry of the parameter set that contains all other parameter sets used to configure that processing step. The Process History Registry contains all the different ordered lists of processing steps that contributed to the file. For example it might contain the list (HLT, RECO) as well as the list (HLT, DEBUG, RECO) because the original file was created by reading two different files that were created using slightly different processing steps. The Production Registry records information about data products. For each data product, it records what labels must be passed to the Event to retrieve that data and the ID in the Parameter Registry of the set of parameters passed to the Producer that created that data. The Parentage Registry holds lists of what data items (recorded as IDs in the Production Registry) were requested from the Event by each Producer.

Then for each event, we record to the file the ID in the Process History Registry of all processing steps that contributed to just this event. We also record the IDs in the Parameter Registry of the sets of parameters that describe which Paths where evaluated when deciding to save the event. The information about the success or failure of the Paths in the event is recorded as a standard data product and is not shown in figure 2. For each data product in the event, we record the ID in the Production Registry which describes this data product. We also record the IDs in the Parentage Registry of the data products that were read by the Producer that created the data.

## 3. Controlling File Size

Of all the types of provenance recorded by the framework, the parental provenance, which is stored per data product per event, can grow the quickest. If unchecked, it can exceed the size of the actual data recorded in the event. Figure 3 shows the recorded relationship between all Producers in an actual RECO file. The colored lower left portion are all the modules from the HLT processing step, while all the modules at the top are from the RECO processing step. These relationships are quite extensive and can in principle change event to event.

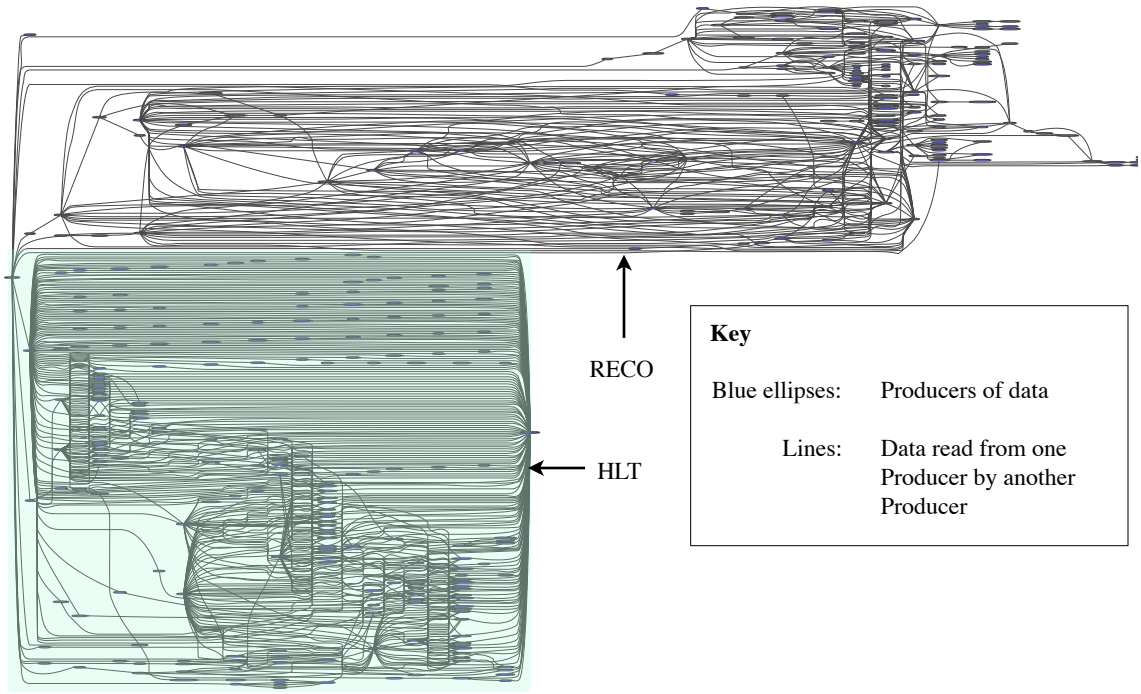


**Figure 3**. Graphical representation of parentage relationship stored in CMS ROOT files.

**Table 1**. Summary of how drop specifications affect which parental provenance is dropped.

| Data origin and storage relation | Drop specification | | | |
|---|---|---|---|---|
| | **None** | **Dropped** | **Prior** | **All** |
| current kept | keep | keep | keep | drop |
| current ancestor | keep | keep | keep | drop |
| current unrelated | drop | drop | drop | drop |
| prior kept | keep | keep | drop | drop |
| prior ancestor | keep | drop | drop | drop |
| prior unrelated | drop | drop | drop | drop |

The framework allows physicists to customize how much parental provenance they wish to keep. We provide four different levels of provenance dropping: none, drop provenance for data that has been dropped, drop provenance for all data coming from a prior processing step, and drop all parental provenance. Table 1 gives a summary of how the different drop specifications determine which parentage provenance is kept, based on different criteria for the event data. The event data criteria is based on what processing step created the data, either the **current** processing step or a **prior** processing step, if the data were explicitly **kept** or if the data were not explicitly kept but were directly or indirectly used to create kept data (i.e. an **ancestor**) or if the unkept data are **unrelated** to any kept data. From the table we see that all drop specifications drop the parental provenance for all unrelated data, because that additional provenance information has no bearing on how any kept data was created. Also the drop specification primarily affects whether provenance information from prior processing steps is copied to the new file. The reason for this is that provenance from prior processing steps can be recovered by re-reading the original files read by the process. In comparison, provenance recorded in the present processing step can never be recovered if it is dropped. CMS's RECO and AOD data sets use the drop specification None, while group or individual physicists skims may use any of the four specifications based on their storage needs.

Table 2 and figure 4 compare the amount of storage in a file taken by provenance for the case of a RECO and an AOD file, as well as a skim that stores only a fraction of the data products per event, where the skim is written using each drop specification. In the skim examples, one could consider that the 1.6% addition to the file size caused by provenance storage is already negligible for the 'none' drop specification. Therefore, additional drop specifications are not necessary. However, in this example, the skim chosen is still fairly heavy weight, at 70kB/event. It would have been fairly easy to

**Table 2**. Comparison of file sizes for different data and drop specifications

| | | | Skimmed data with drop specification | | | |
|---|---|---|---|---|---|---|
| Measurement | RECO | AOD | **None** | **Dropped** | **Prior** | **All** |
| Data/ev (bytes) | 555,386 | 167,054 | 70,092 | 70,092 | 70,092 | 70,092 |
| Prov/ev (bytes) | 4927 | 4743 | 1124 | 58 | 2 | 2 |
| overhead @ 1 GB | 0.9% | 2.9% | 1.6% | 0.12% | 0.04% | 0.04% |

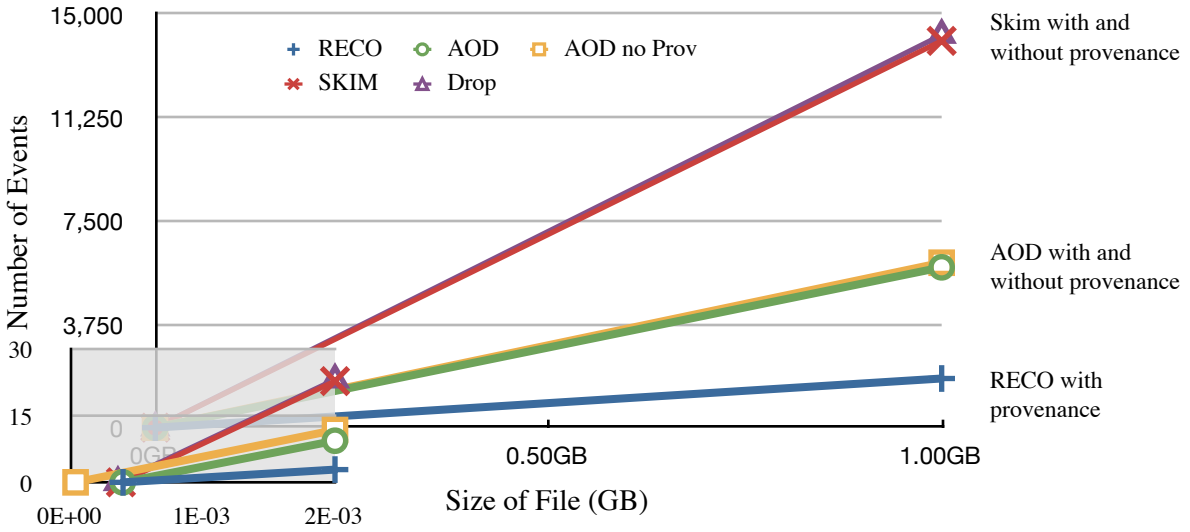


**Figure 4**. Comparison of size of data files to number of stored events for different data groupings and drop specifications

generate a skim where the data size is one tenth of this, i.e. 7kB/event. In that case, the provenance per event would have stayed the same and thus would have been 16% of the file size. Thus the extra drop specifications are primarily used to support very light weight skims.

## 4. Conclusion

CMS's data processing framework records provenance for event selection and data product construction directly into our standard ROOT data files. We have attempted to minimize the disk cost of the provenance storage by allowing physicists to gracefully drop provenance to meet their disk size needs. The provenance storage design has gone through several design and implementation iterations, with at least one more iteration planned based on information uncovered during the study for this paper.

In order for provenance to be useful, it must be used. Framework experts have used the detailed parental provenance information directly from the ROOT files. For example, the information about what modules read which data products was used to do a rough estimate for potential speed improvements should the processing framework be modified to be multi-threaded.

For physicists, we have a first generation tool to inspect the provenance contained in a CMS ROOT file. From such a file, this tool can dump the configuration of each of the Producers along with which data products a Producer created. It can also dump what event selection criteria were used to choose the events in the file. Physicists who have used the tool have given us positive feedback. We plan to expand the tool to include such activities as giving easy access to the per event per product provenance.

## References

[1] Jones C D, Paterno M, Kowalkowski J, Sexton-Kennedy L and Tanenbaum W 2006 *Proc. CHEP 2006* vol 1, ed S Banerjee (India: Macmillan) pp 248-251

[2] Afaq A, Dolgert A, Guo Y, Jones C, Kosyakov S, Kuznetsov V, Lueking L, Riley D and Sekhri V 2008 *J. Phys.: Conf. Ser.* **119** 072001

[3] Brun R and Rademakers F 1997 *Nucl. Inst. & Meth. in Phys. Res.* A **389** 81-86